2009 Fifth International Conference on Autonomic and Autonomous Systems

Predicting web server crashes: A case study in comparing prediction algorithms

Javier Alonso and Jordi Torres
Computer Architecture Department
Technical University of Catalonia
Barcelona Supercomputing Center
alonso@ac.upc.edu, torres@ac.upc.edu

Ricard Gavaldà
Deptartment of Software
Technical University of Catalonia
gavalda@lsi.upc.edu

***Abstract*—Traditionally, performance has been the most important metrics when evaluating a system. However, in the last decades industry and academia have been paying increasing attention to another metric to evaluate servers: availability. A web server may serve many users when running, but if it is out of service too much time, it becomes useless and expensive. The industry has adopted several techniques to improve system availability, yet crashes still happen. In this paper, we propose a new framework to predict time-to-failure when the system is suffering transient failures that consume resources randomly. We study which machine learning algorithms build a more accurate model of the behavior of the anomaly system, and focus on Linear Regression and Decision Tree algorithms. Our preliminary results show that M5P (a Decision Tree algorithm) is the best option to model the behavior of the system under the random injection of memory leaks.**

***Keywords*—Dependability, High-Availability, Prediction Algorithms, Machine Learning.**

I. Introduction

Traditionally, performance has been the most important metrics when evaluating a system. Performance became the most important metric mainly for web application servers because this metric determines the number of clients that our web application server can serve per unit time.

However, in the last decade the industry and academia have been paying increasing attention to other metric to evaluate the servers: availability. A web server that can answer many users when running but if it is out of service too much time becomes useless. For this reason, building high availability servers has become a hot topic in the last years.

This need to overcome the outages of the business and critical systems is clear if we pay attention of the huge loss due the downtime per hour for the industry as exemplified [1]. Moreover, the outages have a negative impact on the company image that could affect indirectly the company profit. To reduce losses and the negative image impact due to the downtime of their services, the industry has adopted several clustering techniques [2] and today most business-critical servers apply some sort of server-redundancy, load-balancers and fail-over techniques. This works quite fine to tolerate application crashes. The latest trend goes towards the development of self-healing techniques [3] that automate recovery procedures and prevent the occurrence of unplanned failures, whenever possible.

The reasons for the downtime of the systems can be divided in three main categories: Human operator errors, software errors, and hardware errors. According to [8] the distribution of errors in these categories is (approximately) 40%, 40% and 20% respectively. If we observe the evolution of this distribution from 1985 to 2005 in detail [4], [5], [6], [7], [8], we see that the proportion of hardware errors has decreased (from 32% in the 80s until 15-20%, nowadays). Likewise, the proportion of operator errors has been stable along time. Although the current systems are growing up in complexity every day, and this fact should in principle involve an increase of human errors during the management tasks, the operators currently have better administration tools to automate some parts of their task. Nevertheless this 40% shows clearly that important challenges are still open in this field.

On the other hand, the software errors have been growing up along the time (from 25% in 80s to 40% today), due mainly to the complexity of the current software and the heterogeneous environment where our systems have to work. According to [4], [5], [7], [8] the most complex failures to overcome are the transient or intermittent failures that cause the resources exhaustion resulting on undesirable behaviour of the system or even hang ups or crashes. Transient failures are difficult to fix because they are difficult or impossible to reproduce when developers try to fix them.

The idea behind this paper is to predict the time until crash. We propose a framework that monitors system-level metrics and predicts the time until crash. We have decided to use system-level metrics because they are available in all systems. In this paper we present the framework we evaluate two different families of Machine learning algorithms: Linear Regression and Decision Trees. Linear Regression has been adopted in several works to model the resource exhaustion. However, in our experiments, Decision Trees seems to adapt better to the behavior of a system under failures due to transient resource exhaustion.

The rest of the paper is as follows: Section II presents the related work around the area. Section III presents briefly Linear Regression and Decision Trees. Section IV describes

978-0-7695-3584-5/09 $25.00 © 2009 IEEE
DOI 10.1109/ICAS.2009.56

the architecture of our framework. Section V presents the experimental environment used to execute the experiments and the results obtained and Section VI presents some conclusions and future work.

II. Related Work

The idea of predicting when a system will exhaust a given resource is not new. A lot of works have modeled this behavior using different machine learning approaches with successful results. In [11] presents an off-line framework to develop performance analysis and post-mortem analysis about the causes of Service Level Objective (SLO) violations. They propose to use TANs (Tree Augmented Naive Bayesian Networks), a simplified version of Bayesian Networks (BN), to determine which resources are most correlated with the performance behavior.

In [10], Linear Regression is used to build an analytic model for capacity planning of multi-tier applications. They show how Linear Regression offers successful results for capacity planning and resource provisioning, even under variable workloads.

Work [9] presents an on-line framework that allows to determinate if our system is suffering an anomaly, workload change or software change. The authors use Linear Regression. The idea is to divide the sequence of recorded data into several segments. A segment is divided in two when no single Linear Regression model gives acceptable error (3%) on the whole segment. If on the two resulting segments there is some model with acceptable error, they determine that we are in front of a software change. If for a performance period it is impossible to obtain a sequence of Linear Regressions with acceptable errors, the conclusion is that the system is suffering some type of anomaly during that period. Their approach is complementary with ours, because the underlying assumption is that, except on transient anomalies and among software changes, the system admits a static model; on the other hand, we concentrate on systems that can degrade, i.e., for which a model valid now will not be valid soon.

The Pinpoint project [13] collects end-to-end traces through the application server with the main goal to determine the more likely component cause of the failures in the system. For this purpose, they use statistical models. The Magpie system [12] collects resource consumptions by components to model with high accuracy the behavior of the system, even distributed ones. However, Magpie and Pinpoint need to instrument the application server. In our approach, we get a less intrusive approach that still guarantees flexibility and adaptability.

III. Classifiers

There are literally hundreds of prediction methods, differing in their capabilities, results, computational costs, and comprehensibility. In this paper we have focused on two types of models: Linear Regression and so-called model decision trees. The reason to choose these two types of models are 1) they allow prediction of continuous, rather than discrete, values, and 2) they are reasonably efficient to build. The first criterion is essential for our purposes, since we want to predict time. The second is not essential at this point, since we are performing offline training, but our immediate next goal will be real-time model building. We have used the implementations in the popular WEKA machine learning package [18], [19].

A. Linear Regression

A linear regression of variables $x_1, \ldots, x_n$ is a function of the form $\alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot x_i$, for some set of coefficients $\alpha_0, \ldots, \alpha_n$. It will work well to predict some variable y when (and only when) y depends linearly on the x_i variables.

B. Decision Trees: REPTree and M5P

For a discrete classification problem, a decision tree is a binary tree where each inner node is labelled with a variable x_i and each leaf is labelled with a class value. In particular, we have used the M5 and REPtree algorithms which use linear regressions at each leave; they are similar and spirit and differ mostly in the method used for pruning unimportant nodes of the tree after it has been built. See the book [19] for more details.

IV. Framework Architecture

In Section IV we present the hypothesis based on our approach to predict the time until failure and the architecture of the framework developed to achieve the goals presented previously.

A. Hypothesis

The main goal of our work is to be able to predict the time until crash using machine learning techniques. To achieve this target, we propose a framework to predict that time. We have developed our framework following the hypothesis that at constant workload and normal behavior, resource consumption is constant, so we want to capture the dynamics of resource consumption. In fact, we have to capture snapshots of the system and evaluate the variation of the resource consumption (its rate or speed). Following that idea, let *CRA* the current amount of a given resource, assume its consumption speed is some constant *RCS*, and that the system crashes when the resource reaches some value *MaxR*. Then the time until crash T must satisfy:

$$Crash = MaxR = RCS * T + CRA, \tag{1}$$

or, isolating T,

$$T = \left(\frac{MaxR}{RCS}\right) - \left(\frac{CRA}{RCS}\right). \tag{2}$$

To help machine learning classifiers deduce that formula by themselves, we have added values *RCS* and *1/RCS* as additional metrics in the dataset. To copmute the variation of the resource consumption along time, we have used the EWMA function [15] on the differences, e.g. *RCS* is EWMA($CRA_t - CRA_{t-1}$).

B. Architecture

The framework is divided in two phases: Training and Test phases. The Training phase has the responsibility to generate the model or classifier to predict the time until crash according to a tuple of system metrics that we obtain from the system. The Test (or prediction) phase computes the time until fault according to the tuple of data describing the current state of the system. Figure 1, shows a diagram of our framework's architecture.

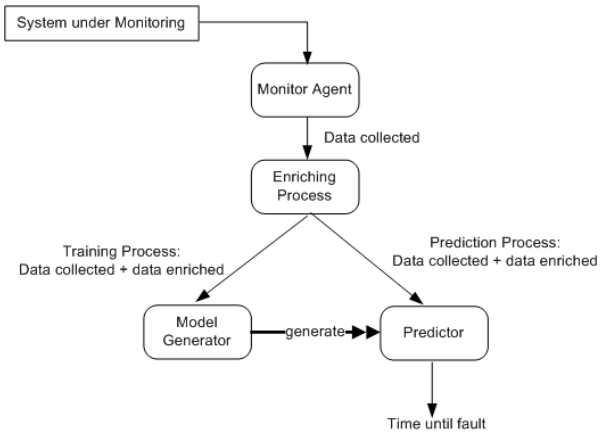

Fig. 1. Detailed Architecture of the Prediction Framework

The *Monitor Agent* is responsible of collecting system metrics. We have decided to collect a small set of metrics that could be collected easily from any operating system. The metrics that we have collected are: Throughput, Response time, workload, System load, disc usage, swap used, number of processes, number of threads, free system memory, memory occupied by the running application/s[1], number of http connections received, and number of connections to the data base. To collect all of this information, we have modified Nagios [14]. Nagios is a well-known and industry-used monitoring tool that allows to execute monitoring gadgets at the system under monitoring every n seconds (in our case, we chose $n = 15$ seconds) and store the results in a file or database.

The architecture of the framework must be thought to eventually run on-line. However, so far we have implemented only an off-line process. More precisely, we collect all the data from one full experiment using the monitoring tool. After that, using an expert, we identify where a crash occurs in the dataset. The new dataset is used as input for the *Enriching process*. The *Enriching process* has the main task to add variables derived from the system metrics. The current version of this module is quite simple: add the EWMA variation described before for any resource metric, because we want to know the mean of the metric but giving more weight to the last metrics to reflect the variation. The idea is to be able to detect potential changes of the consumption of any resource. Furthermore, we add some more expert information used to train. The *Enriching process* adds a column indicating, for each measurement (line) in the dataset, the time until fault at that moment.

After the enriching process is finished, the data set is used to build the model, and then the model is tested using different data sets or using tuple from the same data set but not used for the training process. The flexibility of our framework is that allows to add new machine learning algorithms to build the model, or even to have a set of algorithms to build the model and choose the algorithm according to different parameters, or build a set of models and use them to predict the time until crash and choose the more accurate result, following different strategies like [20], where they have four models and choose the result by general consent (at least 3 of them must agree).

V. Experimental Case Study

After present the reasoning behind our proposal and the framework developed to achieve our goals, we decided to evaluate the effectiveness of the approach running a set of experiments under controlated conditions.

A. Experimental setup

In our experiments, we have used a multi-tier e-commerce site that simulates a on-line book store, following the standard configuration of TPC-W benchmark [16]. We have used the Java version developed using Servlets and using as a Data base server, Mysql [17]. TPC-W allows us to run different experiments using different parameters and under a controlled environment. These capabilities allow us to conduct the evaluation of our approach to predict the time until failure. The details about the machines characteristics is depicted in table I.

TABLE I
Detailed Machines Description

	Clients	Application Server	Data base server
Hardware	2-way Intel XEON 2.4 GHz with 2 GB RAM	4-way Intel XEON 1.4 GHz with 2 GB RAM	2-way Intel XEON 2.4 GHz with 2 GB RAM
Operating System	Linux 2.6.8-3-686	Linux 2.6.15	Linux 2.6.8-2-686
JVM	jdk1.5 with heap 1024MB	jdk1.5 with heap 1024MB	-
Software	TPC-W Clients	Tomcat 5.5.26	MySql 5.0.32

TPC-W clients, called Emulated Browsers (EBs), access to the web site (on-line book store) using the session concept, understanding session like a set of sequenced requests. Between to consecutive requests from the same EB, TPC-W calculates the thinking time, that represents the time while the user receives the web page requested and he decides to request for other more. This value is is generated just before every request and the maximum value could be configured, however in our experiments we have used the default configuration of TPC-W. Moreover, according to the TPC-W, the number of concurrent EBs is constant during the experiment.

[1] it will be Tomcat, in our experiments

To simulate a transient failure that consumes resources until their exhaustion, we have modified a servlet of the TPC-W implementation. In our case, we have modified the *TPCW_search_request_servlet* class. This class calculates a random number between 0..N (in our experiments N=30). This number determinates how many clients have to request the Servlet before to inject the memory leak. This behavior makes that the variation of memory consumption depends of the number of clients and the frequency of servlet visits. According to the TPC-W specification, this frequency depends on the workload chosen. TPC-W has three types of workload (browsing, Shopping and Ordering). In our case, we have conducted all of our experiments using shopping distribution. The EBs in this workload visit the Servlet modified around 20%. This makes that with high workload our servlet injects quickly memory leaks, however with low workload, the consumption is lower too. The memory leak injected in our experiments is fixed (1MB).

B. Experimental Results

1) Linear Regression vs REPTree: We want to evaluate which of the chosen machine learning algorithms were more effective and accurate to predict the time until failure. For this reason, we have conducted experiments with different workloads to train and test the model built. First, we conducted a set of experiments with 5, 25, 50, 100 and 200 EBs until the system crashes, and collected the system metrics described in Section IV. In our experiments we define a crash when the server is unavailable to answer any request during 30 seconds.

In Figure 2 we present the Tomcat memory consumption with a workload of 50EBs. We can see two flat phases around 6700 seconds before crash and around 1885 seconds before crash. These two flat phases are due to the action of the Garbage Collector (GC) and the JVM heap management. At these two points, the Old section of the heap has to be resized to allow adding new objects, and that action provokes that the GC could free as memory as the memory injector injects. We can see that there is an almost linear relationship

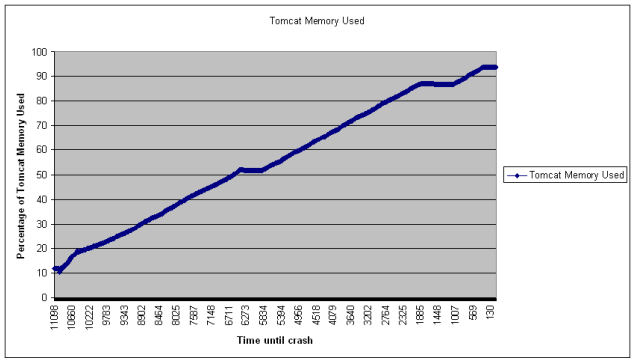

Fig. 2. Memory Consumption with 50EBs injecting random leaks

between Tomcat memory consumption and the time until failure. However, if we observe the rate as our fault injector is injecting the memory leaks, it is not. Figure 3 shows the rate of Tomcat memory consumption along the experiment, we can see that the memory consumption is quite irregular. Furthermore, we can see how the flat phases presented before are clearly visible in the graph, where the variation (speed) of resource consumption goes down.

This behavior has been observed in all experiments conducted (with 5, 25, 50, 100 and 200 EBs), but we do not present all of them. After running all of these experiments and passing the enriching process, we used these experiments to build the model and make an initial test, using a percentage split method: a random selection of 66% of the data set is used to train the model and the remaining 34% of the same data set for testing.

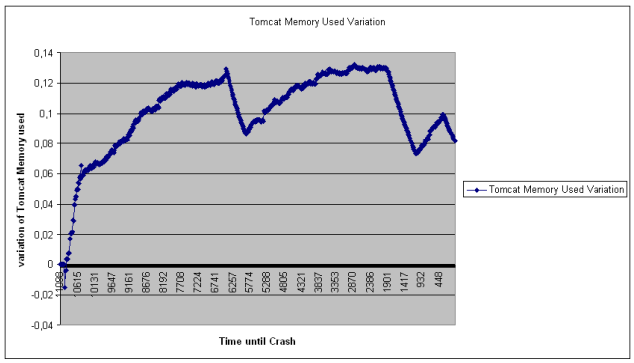

Fig. 3. Comparison between raw and filtered data for memory consumption rate, with 50EBs

After that, we conducted two experiments more with 75 and 150EBs. The idea was to use these two datasets to demonstrate the accuracy achieved by the models generated and their generalization ability. We used the same data sets (both training and testing data sets) for both algorithms, Linear Regression and REPTree (both of them from Weka 3.5.8 [18]).

The detailed results using percentage split and using the two external experiments are presented in Table II. It shows clearly that the Decision Tree method predicts better in this context: REPTree algorithm predicts time-until-crash with an average error of minutes, while Linear Regression obtains an unacceptable average error (around hours of error, at times when the system predicts a time until crash of 10 hours later, yet the crash happens 10 minutes later). We next provide our interpretation for this fact.

Linear Regressions are adequate when the variable to be predicted depends linearly on the input variables; this is roughly true when systems are in relatively stable states, and not under stress. This is why Linear Regression was successfully used, since they were modelling the system at stable states within anomalies [9]. The authors assumed the fact that the resource consumption has a relationship with the workload. However, as the system approaches resource exhaustion, complicated, highly nonlinear effects appear: for example, when the load is already quite high, any slight increase in load creates thrashing, and the system collapses. A small % in load can increase system response time from

TABLE II
MEAN ERROR COMPARISON BETWEEN LINEAR REGRESSION AND REPTREE ALGORITHMS FOR TIME UNTIL FAILURE(RAW DATA AND FILTERED DATA)

	Percentage Split (66%)	Supplied Test (workload 75EBs)	Supplied Test (workload 150EBs)
Linear Regression	9861.25 sec	12420.65 sec	6062321.22 sec
RepTree	908.28 sec	1429.16 sec	1147.66 sec
Linear Regression filtered data	13248.82 sec	7999.87 sec	23740.37 sec
REPTree filtered data	644.52 sec	1816.01 sec	468.90 sec

0.1 second to virtually infinite. Since we are interested in modelling the system near crashes, we must be able to model such highly nonlinear effects, and Decision Trees do this by splitting the input space in regions with completely different behaviors. Additionally, Linear Regression does not deal well with outliers: for example, from the point of view of computing the average rate of memory consumption, the precise moment at which a leak occurs will create a transient peak, that not even our EWMA smoother can completely avoid.

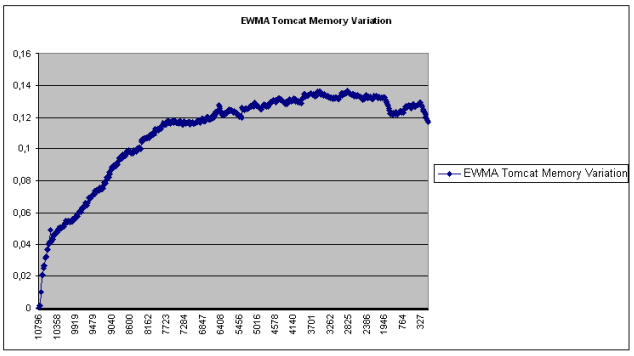

Fig. 4. Variation of the rate of Memory Consumption with 50EBs injecting memory leaks randomly (filtered data)

Another important point is the fact that we are using the raw data without removing warm up periods or noise introduced during the resizing the heap Old section by the GC. If we remove warm up and flat phases (filtered data) during the execution we obtain different results (showed in table II). We can see a general improvement in both cases, but Linear Regression still obtains worse results. However, we observe a strange behavior in REPTree with 75EBs using the model generated with filtered data. We obtain worst results because the problem here is that data from 5 and 25EBs introduces memory leaks very slowly (yet randomly), so the estimation of the memory consumption rate is quite unstable, and extrapolation to 75EBs not very reliable. With 150EBs, the model obtains better the results because the behavior of the memory leak in front of high workloads (100 and 200EBs) is more stable.

2) Merging the Best Capabilities: M5P: Finally, if we observe figure 4 where we present the variation of the memory consumption without warm up and plain phases of variation (data filtered) for 50EBs, we observed that we can see the plot like several of little linear regressions. Following this idea we decide to use M5P algorithm. This algorithm is interesting because it has the better from Linear Regression and the better of Decision Trees. In fact, every leaf of the decision tree contains a Linear Regression model.

In Table III we observe the results using M5P using raw data (no warmup or GC activity periods removed). All of them demonstrate that our intuition was in the correct direction, because prediction errors are consistently lower than those obtained by RepTree or Linear Regression. The 75EBs experiment is still the worst case. Still, we see that using M5P predicts the time to crash with an average error of about 10 minutes, allowing for the possibility of applying self-healing techniques or alerting human operators.

TABLE III
AVERAGE ERROR COMPARISON BETWEEN LINEAR REGRESSION AND REPTREE ALGORITHMS FOR TIME UNTIL FAILURE

	Percentage Split (66%)	Supplied Test (workload 75EBs)	Supplied Test (workload 150EBs)
M5P Raw data	562.85 sec	1702.31 sec	654.80 sec
R5P with filtered data	506.08 sec	989.10 sec	402.71 sec

Taking another point of view, in our last experiment we evaluated sequentially, tuple by tuple the predicted time and the real time until fault. The idea is to check whether errors occur uniformly along time or whether they tend to occur far or near failures. Figure 5 shows the time until failure predicted by M5P using raw data and filtered data in this sequential setting, using as a test data set the 150EBs experiment.

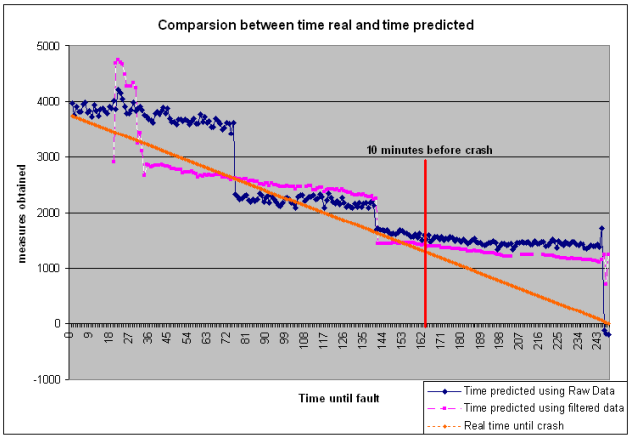

Fig. 5. Accuracy of the model using M5P and tested with 150EBs data set

We can observe how the model has an acceptable accuracy to predict the time until fault during the first part of the execution, however, as we get closer to the actual crash, and

especially in the last 10 minutes, the model starts to be poor accurate to predict the time. We see thus that our approach has to be improved to be more accurate in the last part, because it is then when the model should predict better, to avoid false positives or apply corrective techniques when they are really needed.

VI. Conclusion and Future work

In this paper we have presented an evaluation of two types of machine learning algorithms when used to model and predict resource consumption due to transient or intermittent failure. We can conclude that although Linear Regression has been used in several studies to model the behavior of the resource usage with successful results, it seems to fail in our setting to predict the behavior of the system under anomalies. By contrast, we conclude that Decision Trees seem like a good option to model anomalies. This does not exclude, of course, that other machine learning algorithms perform even better. We believe that support vector machines are very promising with respect to accuracy, but they tend to be computationally expensive and our final goal is to work on-line and in real time; still, it is in our plans to test them. This study is complementary to works like [9], [10] or [11], because the framework can use Linear Regression when the system is running without anomalies, but when an anomaly is detected, our framework could be used to determine the risk of system.

On the other hand, we have to refine the M5P algorithm to try to fix the poor accuracy obtained in the last period of the experiments, i.e., near the crash. Our idea is to give additional weight to the tuples from the last period (say, 10 minutes before the crash) during the training phase. The idea is to force M5P to pay more attention to (i.e., fit better) the most important part of the data during the training phase. Finally, we want merge this predictive framework with some recovery techniques like VM-Rejuv [21]. VM-Rejuv is a self-healing framework focused on software aging problem that triggers the recovery action (software rejuvenation) based on simple thresholds. Our future goal will be to demonstrate that a predictive approach is more effective than a simple threshold to decide when is the moment to apply the recovery actions.

VII. Acknowledgements

This research work is carried out in part under the FP6 Network of Excellence Core-GRID funded by the European Commission (Contract IST-2002-004265). Also this work has been supported by the Spanish Ministry of Education and Science (projects TIN2007-60625). R. Gavaldà is partially supported by the EU PASCAL2 Network of Excellence and by the MICINN SESAAME project, TIN2008-06582-C03-01/TIN.

References

[1] J.Hennessy and D.Patterson. *Computer Architecture: A Quantitative Approach*. Morgan & Kaufmann Publishers, 2002.

[2] E.Marcus and H.Stern. *Blueprints for High Availability*. Wiley, 2003

[3] J.Kephart and D.M.Chess. *The Vision of Autonomic Computing*. IEEE Computer, 36(1), 2003

[4] J. Gray *Why do computers stop and what can be done about it?*. 5th SRDS, Jan. 1986.

[5] J. Gray *A census of Tandem system availability between 1985 and 1990*. Tandem Computers Technical Report 90.1, 1990.

[6] D. Patterson et al. *Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*, tech. report CSD-02-1175, Computer Science Dept., Univ. of Calif., Berkeley, Calif., Mar. 2002.

[7] D. Oppenheimer, A. Ganapathi, and D. Patterson *Why do Internet services fail, and what can be done about it?*. in 4th USENIX Symp. on Internet Technologies and Systems (USITS03), 2003.

[8] Soila Pertet and Priya Narasimham *Causes of Failure in Web Applications*. Tech. report CMU-PDL-05-109, Carnegie Mellon University, Dec. 2005.

[9] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni *Anomaly? Application Change? or Workload Change? Towards Automated Detection of Application Performance Anomaly and Change*. Procs. 38th Annual IEEE/IFIP Conf. on Dependable Systems and Networks, DSN'2008, June 24-27, 2008.

[10] Q. Zhang, L. Cherkasova, N. Mi, and E. Smirni *A regression-based analytic model for capacity planning of multi-tier applications*. Cluster Computing (2008), vol 11: 197-211.

[11] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase *Correlating instrumentation data to system states: A building block for automated diagnosis and control*. In Proc. 6th USENIX OSDI, San Francisco, CA, Dec. 2004.

[12] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier *Using Magpie for request extraction and workload modelling*. Proc. of the 6th Symp. OSDI'2004.

[13] M. Chen, A. Accardi, E. kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. *Path-based failure and evolution management*. Proc. of the 1st Symp. NSDI'2004.

[14] Nagios Monitoring Tool *http://www.nagios.org/*. 02.21.2009.

[15] S. W. Roberts *Control-charts-tests based on geometric moving averages*. Technometrics 1: 239-250. 1959.

[16] TPC-W Java Version *http://www.ece.wisc.edu/ pharm/*. 02.21.2009.

[17] MySQL Data Base server *http://www.mysql.com/*. 02.21.2009.

[18] Weka 3.5.8 *http://www.cs.waikato.ac.nz/ml/weka/*. 02.21.2009.

[19] I. H. Witten and E. Frank *Data Mining: Practical Machine Learning Tools and Techniques (2nd ed.)*. Morgan Kaufmann, 2005.

[20] G. Yoo, J. Park, and E. Lee *Hybrid Prediction Model for improving Reliability in Self-healing System*. Proc. 4th Int. Conf. on Software Engineering Research, Management and Applications (SERA'06), 2006.

[21] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak *Using virtualization to improve software rejuvenation*. In Proc. of the 6th IEEE Int. Symp. on Network Computing and Applications (NCA'2007), pages 33-44, 2007.